2100030336

M. Gayathri Ankitha

.NET PROGRAMMING

LAB 5

IN-LAB

TASK1: To create classes **Deposit** (bank account), **BaseDeposit** (regular deposit), **SpecialDeposit** (special deposit), **LongDeposit** (long-term deposit), **Client** (bank client) with set functionality.

1. To create abstract class **Deposit** and declare within it:

- Public money property only for reading **Amount** (deposit amount)
- Public integer property only for reading **Period** (time of deposit in months)
- Constructor (for calling in class-inheritor) with parameters **depositAmount** and **depositPeriod**, which creates object deposit with specified sum for specified period.
- Abstract method **Income**, which returns money value – amount of income from deposit. Income is the difference between sum, withdrawn from deposit upon expiration date and deposited sum.

2. To create classes that are inheritors of the class **Deposit**, which determine different options of deposit interest addition – class **BaseDeposit**, class **SpecialDeposit** and class **LongDeposit**. To implement in each class a constructor with parameters **amount** and **period**, which calls constructor of parent class.
3. For each inheritor class – to implement own interest addition scheme and accordingly profit margin definitions, overriding abstract method **Income** in each class.

BaseDeposit implies each month 5% of interest from current deposit sum. Each following month of income is calculated from the sum, which was received by adding to current income sum of the previous month and is rounded to hundredth.

Example: Base amount – 1000,00

In a month – 105,00; income amount – 50,00

In two months – 1102,50; income amount – 102,50

In three months – 1157,62; income amount – 157,62

SpecialDeposit implies income addition each month, amount of which (in percent) equals to deposit expiration period. If during the first month 1% is added, during the second month – 2% from the sum obtained after first month and so on.

Example: Base amount – 1000,00

In a month – 1010,00; income amount – 10,00

In two months – 1030,20; income amount – 30,20

LongDeposit implies that during first 6 months, no percent is added to client's deposit, but starting from 7th month, each month percent addition is 15% from current deposit sum, thus encouraging client to make long-term deposits.

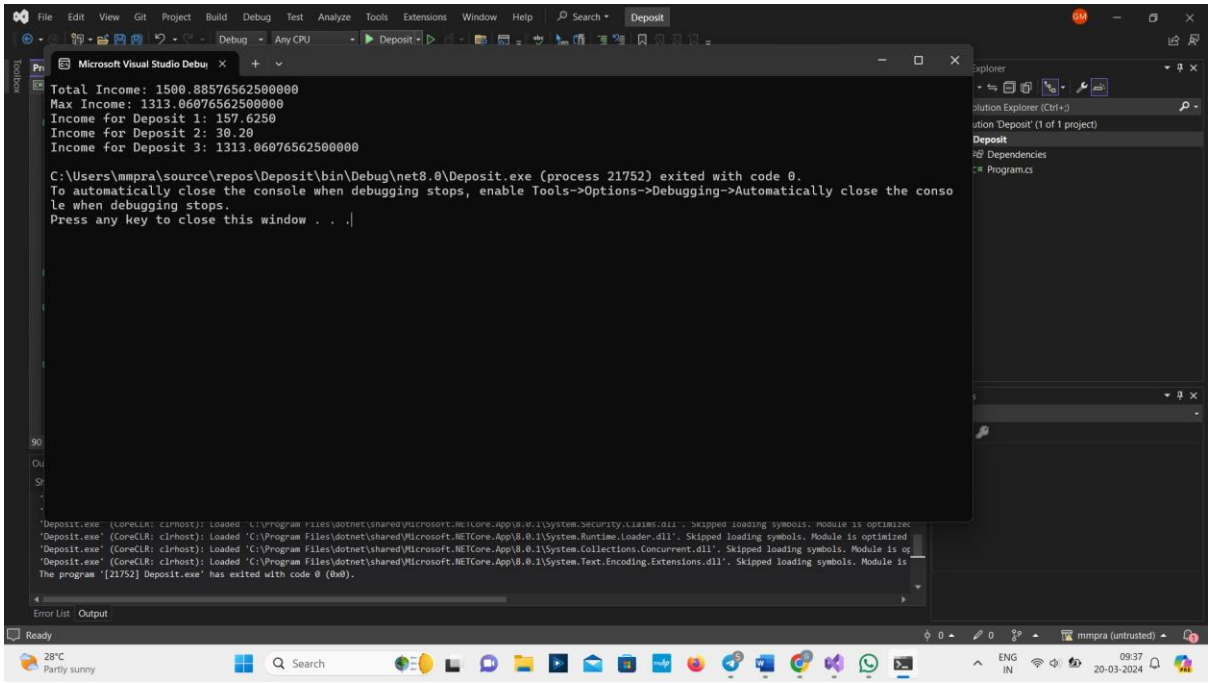
4. To create class **Client** (bank client) and declare within it:

- Private field **deposits** (client deposits) – objects array of type Deposit
- Constructor without parameters, which creates empty array deposits consisting of 10 elements
- Method **AddDeposit** with parameter **deposit** for adding regular, special or long-term account into array on the first empty spot and returning true, or returning false, if accounts number limit is depleted (no empty space in array).
- Method **TotalIncome**, returning total income amount based on all client's deposits upon deposits expiration.
- Method **MaxIncome**, returning maximum deposit income of all client's deposits upon deposits expiration.
- Method **GetIncomeByNumber** with integer parameter **number** (deposit number, which equals its index in array, increased by one), returning income from deposit with such number. If deposit with such number does not exist, method returns 0 value.

```
using System;
public abstract class Deposit
{
    public decimal Amount { get; }
    public int Period { get; }
    public Deposit(decimal depositAmount, int depositPeriod)
    {
        Amount = depositAmount;
        Period = depositPeriod;
    }
    public abstract decimal Income();
}
public class BaseDeposit : Deposit
{
    public BaseDeposit(decimal amount, int period) : base(amount, period) { }
    public override decimal Income()
    {
        decimal totalIncome = 0;
        decimal currentAmount = Amount;
        for (int i = 0; i < Period; i++)
        {
            decimal monthlyIncome = currentAmount * 0.05m;
            totalIncome += monthlyIncome;
            currentAmount += monthlyIncome;
            currentAmount = Math.Round(currentAmount, 2);
        }
        return totalIncome;
```

```
    }
}
public class SpecialDeposit : Deposit
{
    public SpecialDeposit(decimal amount, int period) : base(amount, period) { }
    public override decimal Income()
    {
        decimal totalIncome = 0;
        decimal currentAmount = Amount;
        for (int i = 1; i <= Period; i++)
        {
            decimal monthlyIncome = currentAmount * (decimal)i / 100;
            totalIncome += monthlyIncome;
            currentAmount += monthlyIncome;
        }
        return totalIncome;
    }
}
public class LongDeposit : Deposit
{
    public LongDeposit(decimal amount, int period) : base(amount, period) { }
    public override decimal Income()
    {
        decimal totalIncome = 0;
        decimal currentAmount = Amount;
        for (int i = 1; i <= Period; i++)
        {
            decimal monthlyIncome = (i <= 6) ? 0 : currentAmount * 0.15m;
            totalIncome += monthlyIncome;
            currentAmount += monthlyIncome;
        }
        return totalIncome;
    }
}
public class Client
{
    private Deposit[] deposits;
    public Client()
    {
        deposits = new Deposit[10];
    }
    public bool AddDeposit(Deposit deposit)
    {
        for (int i = 0; i < deposits.Length; i++)
        {
            if (deposits[i] == null)
            {
                deposits[i] = deposit;
                return true;
            }
        }
        return false; // Array full
    }
    public decimal TotalIncome()
    {
        decimal totalIncome = 0;
        foreach (var deposit in deposits)
        {
            if (deposit != null)
                totalIncome += deposit.Income();
        }
        return totalIncome;
    }
```

```
    public decimal MaxIncome()
    {
        decimal maxIncome = 0;
        foreach (var deposit in deposits)
        {
            if (deposit != null)
            {
                decimal income = deposit.Income();
                if (income > maxIncome)
                    maxIncome = income;
            }
        }
        return maxIncome;
    }
    public decimal GetIncomeByNumber(int number)
    {
        if (number < 1 || number > deposits.Length || deposits[number - 1] ==
null)
            return 0;
        return deposits[number - 1].Income();
    }
}
class Program
{
    static void Main(string[] args)
    {
        Client client = new Client();
        client.AddDeposit(new BaseDeposit(1000.00m, 3));
        client.AddDeposit(new SpecialDeposit(1000.00m, 2));
        client.AddDeposit(new LongDeposit(1000.00m, 12));
        Console.WriteLine("Total Income: " + client.TotalIncome());
        Console.WriteLine("Max Income: " + client.MaxIncome());
        Console.WriteLine("Income for Deposit 1: " +
client.GetIncomeByNumber(1));
        Console.WriteLine("Income for Deposit 2: " +
client.GetIncomeByNumber(2));
        Console.WriteLine("Income for Deposit 3: " +
client.GetIncomeByNumber(3));
    }
}
```



TASK 2: To add the following new functionalities to the project created in task Aggregation:

1. To create interface **Iprolongable** (prolonging deposit) and declare within it method **CanToProlong** without parameters that returns logic value true or false, depending on the fact whether this specific deposit can be prolonged or not.
2. To implement interface **IProlongable** in classes **SpecialDeposit** and **LongDeposit**.
3. In addition, special deposit (**SpecialDeposit**) can be prolonged only when more than 1000 UAH were deposited, and long-term deposit (**LongDeposit**) can be prolonged if the period of deposit is no longer than 3 years.
4. To implement standard generic interface **IComparable<Deposit>** in abstract class **Deposit**. Total sum amount (sum deposited plus interest during entire period) should be considered as comparison criteria of **Deposit** instances.
5. To implement additionally in class **Client**:

- interface **IEnumerable<Deposit>**.
- Method **SortDeposits**, which performs deposits sorting in array **deposits** in descending order of total sum amount on deposit upon deposit expiration.
- Method **CountPossibleToProlongDeposit**, which returns integer – amount of current client's deposits that can be prolonged.

```
using System;
using System.Collections;
```

```
using System.Collections.Generic;
public interface IProlongable
{
    bool CanToProlong();
}
public abstract class Deposit : IComparable<Deposit>
{
    public decimal Amount { get; }
    public int Period { get; }
    public Deposit(decimal depositAmount, int depositPeriod)
    {
        Amount = depositAmount;
        Period = depositPeriod;
    }
    public abstract decimal Income();
    public decimal TotalAmount()
    {
        decimal totalInterest = Income();
        return Amount + totalInterest;
    }
    public int CompareTo(Deposit other)
    {
        return TotalAmount().CompareTo(other.TotalAmount());
    }
}
public class SpecialDeposit : Deposit, IProlongable
{
    public SpecialDeposit(decimal amount, int period) : base(amount, period) { }

    public bool CanToProlong()
    {
        return Amount > 1000;
    }
    public override decimal Income()
    {
        return 0;
    }
}
public class LongDeposit : Deposit, IProlongable
{
    public LongDeposit(decimal amount, int period) : base(amount, period) { }
    public bool CanToProlong()
    {
        return Period <= 36;
    }
    public override decimal Income()
    {
        return 0;
    }
}
public class Client : IEnumerable<Deposit>
{
    private List<Deposit> deposits;
    public Client()
    {
        deposits = new List<Deposit>();
    }
    public bool AddDeposit(Deposit deposit)
    {
        deposits.Add(deposit);
        return true;
    }
    public int CountPossibleToProlongDeposit()
```

```
    {
        int count = 0;
        foreach (var deposit in deposits)
        {
            if (deposit is IProlongable &&
((IProlongable)deposit).CanToProlong())
                count++;
        }
        return count;
    }
    public void SortDeposits()
    {
        deposits.Sort();
    }
    public IEnumerator<Deposit> GetEnumerator()
    {
        return deposits.GetEnumerator();
    }
    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}
class Program
{
    static void Main(string[] args)
    {
        Client client = new Client();
        client.AddDeposit(new SpecialDeposit(1500, 12));
        client.AddDeposit(new LongDeposit(2000, 24));
        Console.WriteLine("Number of deposits possible to prolong: " +
client.CountPossibleToProlongDeposit());
    }
}
```